Available online at www.sciencedirect.com
ScienceDirect
Journal of Systems Architecture 53 (2007) 644–658
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A multi-channel architecture for high-performance NAND flash-based storage system ☆

Jeong-Uk Kang [a,*], Jin-Soo Kim [a], Chanik Park [b], Hyoungjun Park [b], Joonwon Lee [a]

[a] Division of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea
[b] Memory division, Semiconductor Business, Samsung Electronics Co., Republic of Korea

Received 9 December 2005; received in revised form 5 January 2007; accepted 8 January 2007
Available online 1 February 2007

Abstract

Many mobile devices demand a large-capacity and high-performance storage system in order to store, retrieve, and process large multimedia data quickly. In this paper, we present a high-performance NAND flash-based storage system based on a multi-channel architecture. The proposed system consists of multiple independent channels, where each channel has multiple NAND flash memory chips. On this hardware, we investigate three optimization techniques to exploit I/O parallelism: striping, interleaving, and pipelining. By combining all the optimization techniques carefully, our system has shown 3.6 times higher overall performance compared to the conventional single-channel architecture.

Keywords: NAND flash memory; Storage system; I/O parallelism

1. Introduction

Flash memory is widely used for code and data storage of consumer electronics products due to its versatile features such as non-volatility, solid-state reliability, low power consumption, and shock resistance [5]. The most popular flash memory types are NOR and NAND. Although NOR flash memory offers random access capability and high read performance, it suffers from extremely low write and erase performance and it is more expensive per MB than NAND flash memory. On the other hand, NAND flash memory provides high cell densities and low cost per MB with higher write and erase performance than NOR flash memory. Therefore, NOR flash memory is well suited for code storage and execute-in-place (XIP) applications, while NAND flash memory is suitable for data storage [13].

Many mobile devices, including MP3 players, PDAs (personal digital assistants), PMPs (portable media players), high-resolution digital cameras and

☆ This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Inofrmation Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment) (IITA-2006-C1090-0603-0020).
* Corresponding author. Tel.: +82 42 869 3585; fax: +82 42 869 5569.
E-mail address: ux@calab.kaist.ac.kr (J.-U. Kang).

1383-7621/$ - see front matter
doi:10.1016/j.sysarc.2007.01.010

camcorders, and mobile phones, demand a large-capacity and high-performance storage system in order to store, retrieve, and process large multimedia data quickly. In those devices, NAND flash memory is already becoming one of the most common storage medium. Moreover, solid-state flash disks based on NAND flash memory technology, such as M-System's FFDs (Fast Flash Disks) [6] and BiTMICRO's E-Disks [7], are gradually replacing mechanical hard disks under mission-critical and/or rugged operating conditions in military and aerospace industry. As NAND flash technology development continues to double density growth on an average of every 12 months [16], it is expected that sub-notebook computers or tablet PCs equipped with more than tens of Gbytes of NAND flash memory-based storage system will be available to ordinary users in the near future.

Although replacing hard disks with NAND flash memory brings advantages in terms of size, weight, reliability, and energy use, it is not easy to draw the maximum performance from NAND flash memory due to its unique operational characteristics. In NAND flash memory, the write operation requires a relatively long latency compared to the read operation. In addition, the previous data should be erased first in order to write another data in the same physical area. The worse problem is that the erase operation cannot be performed on the particular data selectively, but on the larger unit containing the original data with much longer latency. The MLC (Multi-Level Cell) technology, which is recently introduced to multiply the capacity of a NAND flash memory chip, further decreases the operation speed [1]. Thus, developing a high-performance NAND flash-based storage system remains a technically challenging area.

In this paper, we propose a hardware and software architecture of high-performance NAND flash-based storage system. Specifically, we focus on three optimization techniques that can exploit I/O parallelism in various ways: *striping*, *interleaving*, and *pipelining*. The hardware architecture of the proposed system consists of multiple independent channels, where each channel has multiple NAND flash memory chips. On this hardware, we quantitatively investigate the performance impact of individual optimization technique, as well as the overall performance when all the techniques are combined together.

In the experiments, the striping technique, the interleaving technique, and the pipelining technique improve the throughput up to 164%, 72%, and 197%, respectively. By combining all the optimization techniques carefully, our system has shown 3.6 times higher overall performance compared to the conventional single-channel architecture.

The rest of the paper is organized as follows. Section 2 describes the characteristics of NAND flash memory. In Section 3, we present the hardware architecture of the high-performance NAND flash-based storage system that we designed and implemented. In Section 4, we explain the software architecture and three optimization techniques that can maximize I/O parallelism. Section 5 shows experimental results and Section 6 presents the related work. Finally, our conclusions and future work are drawn in Section 7.

2. Background

A NAND flash memory consists of a memory array and an I/O buffer. Data are transferred through the I/O buffer. The memory array is composed of a fixed number of blocks and each block is organized as 32 pages. A *block* is a basic unit of erase operations, while a *page* is a basic unit of read and write operations. Each page consists of 512 bytes of main area and 16 bytes of spare area.

There are three basic operations in NAND flash memory: read, program (write), and erase. The read operation fetches data from a target page, while the program operation writes data to a target page. The erase operation resets all values of a target block to 1. In flash memory, once a page is written, it should be erased before the subsequent program operation is performed on the same page. This characteristic is sometimes called erase-before-write. The number of program/erase cycles is limited to about 10,000–1,000,000 times.

Read and program operations consist of three phases: setup, busy, and data transfer. In order to read a page, command and address are given to a NAND flash memory chip through I/O pins in the setup phase. After a fixed delay time of 10–25 μs, the selected page is loaded into the I/O buffer in the busy phase. Hereafter 8-bit or 16-bit data can be sequentially fetched from the I/O buffer every 50 μs in the data transfer phase. The program operation is similar to read, except that the sequence of the data transfer phase and the busy phase is reversed. The programming delay time is 200–700 μs. This fixed delay time of program operation is about 10 times longer than that of read operation.

The erase operation is simply composed of the setup phase followed by the busy phase, since data transfer is not needed. The fixed delay time of erase operation in the busy phase is 2–3 ms [14].

Recently, a new type of NAND flash memory, called *large block NAND*, has been introduced in order to provide high density and high performance in bulk data transfer. The large block NAND flash memory offers 2 Kbytes of main area and 64 bytes of spare area with 64 pages per block. Note that a new programming restriction is added in the large block NAND flash memory; the page should be programmed in sequential order from page 0 to page 63 within a block. Random page address programming in a block is strictly prohibited by the specification [14].

3. Multi-channel architecture

In this section, we describe the hardware architecture of our prototype system, which is based on a multi-channel architecture. The basic idea behind the multi-channel architecture is to exploit I/O parallelism by utilizing multiple independent channels and multiple NAND flash memory chips.

3.1. Overall architecture

In order to investigate the impact of the multi-channel architecture on the overall performance, we implemented a prototype of NAND flash-based embedded storage system, which we named DUMBO. The design of DUMBO focuses on simplicity and flexibility so that we can investigate the effect of various optimization techniques.

Fig. 1 depicts the overall architecture of the prototype system where DUMBO is connected to the host system through the host interface. DUMBO consists of four independent channel managers that control NAND flash memory chips separately.

The host system is an ARM9-based embedded system which has 32 Mbytes SDRAM and 64 Mbytes NOR flash memory. The clock speed of CPU is 100 MHz. Operating system and read only data are stored in the NOR flash memory. The host system has an additional SDRAM interface for external I/O devices. We use this additional SDRAM interface to connect DUMBO to the host system. The bus width of the SDRAM interface is 32 bits and the maximum bandwidth is 25 Mbytes/s. To support interrupt-driven I/O, an interrupt line is connected to the host CPU.

3.2. Channel manager

Each channel manager consists of a control logic, a NAND interface, two independent buffers, and eight NAND flash memory chips as illustrated in Fig. 2. The control logic is responsible for transferring data between NAND flash chips and the corresponding channel manager's buffers. The NAND interface controls NAND flash memory chips to read and write data. The control logic receives read and write commands from the host system through the host interface, and manages NAND flash memory chips via the NAND interface to service the commands. When the data transfer finishes, the control logic notifies the host interface of the completion of the command. Then the host interface sends an interrupt signal to the host system.

We used the large block type of NAND flash memory chips for DUMBO. The capacity of each NAND flash memory chip is 1 Gbits. Thus, each channel manager provides 1 Gbytes and the total storage size of DUMBO is 4 Gbytes. NAND flash memory chips are connected to buffers via 16-bit

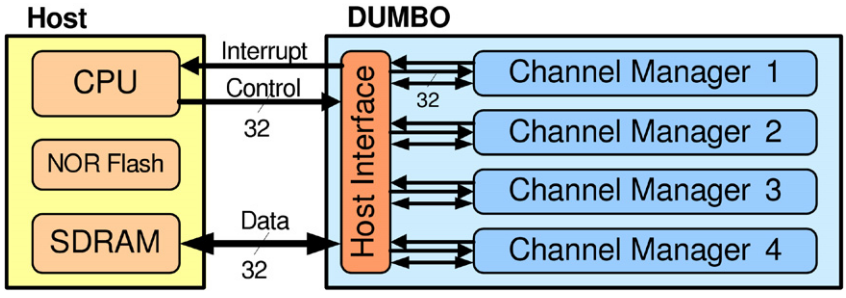

Fig. 1. The overall architecture of the prototype system.

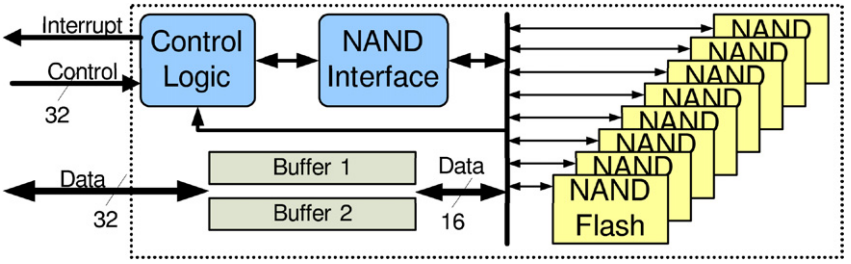

Fig. 2. The internal organization of a channel manager in DUMBO.

I/O pins. Since the buffers are accessed both from the host interface and from the NAND interface, they are implemented with dual-port RAMs. The size of each buffer is 2112 bytes that is the same as the page size (including the 64-byte spare area) of the large block NAND flash memory.

As each channel manager has two independent buffers, we can send the next command to the same channel manager using the other buffer without waiting for the completion of the previous command. All the eight NAND flash memory chips in the same channel manager share address/data I/O lines. Therefore, we cannot transfer data between two buffers and several NAND flash memory chips concurrently. We can, however, overlap several operations using separate control lines and two independent buffers in the channel manager. The details will be given in Section 4.

3.3. Read operation

A read operation consists of three phases: Read Set (*RS*), Read from NAND (*RN*), and Read Data (*RD*) as shown in Fig. 3. The arrows in the boxes indicate the flow of control information and data.

In order to read a page, read control information is given to DUMBO via the host interface (*RS*). Then the control logic of the channel manager controls a NAND flash memory chip to read data. Read command and page address are given to the NAND flash memory chip via the NAND interface (setup phase). After the busy phase, the control logic moves data to the buffer (data transfer phase). When the data transfer from the NAND flash memory chip to DUMBO (*RN*) finishes, DUMBO sends an interrupt signal to the host system to notify the completion of the request. Finally, the host system copies data from DUMBO (*RD*).

In Fig. 3, the boxes with diagonal lines (*RS* and *RD*) use the host interface and the boxes with horizontal lines (*RN*) use the NAND interface. Although there is no data transfer in the busy phase, the channel manager polls the NAND flash memory chip in order to check the completion status using the NAND interface. Polling during the busy phase is more efficient than using an interrupt-driven I/O because the delay time of the busy phase is very short.

3.4. Write operation

A write operation consists of five phases: Write Data (*WD*), Write Set (*WS*), Write to NAND (*WN*), Write Confirm (*WC*), and NAND Program (*NP*) as illustrated in Fig. 4. Note that the boxes with diagonal lines (*WD*, *WS*, and *WC*) use the host interface and the boxes with horizontal lines (*WN*) use the NAND interface. The box with diagonal crossing lines (*NP*) does not use any interfaces.

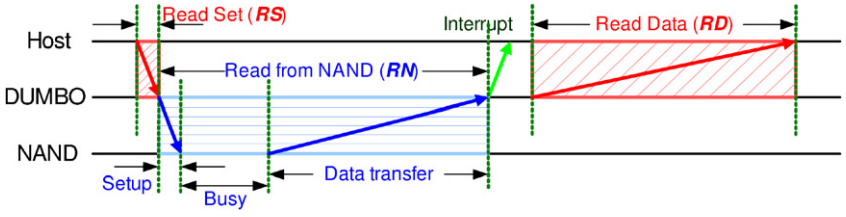

Fig. 3. Timing diagram of read operation in DUMBO.

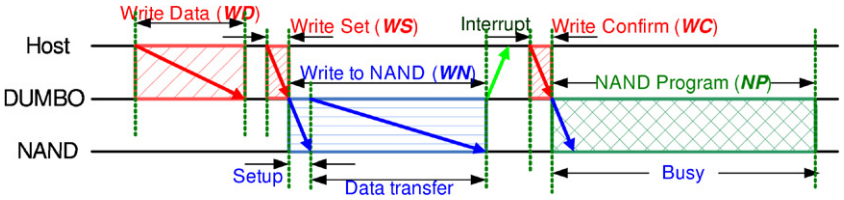

Fig. 4. Timing diagram of write operation in DUMBO.

In order to write a page, the host system moves data to the buffer of the corresponding channel manager in DUMBO (*WD*) and sends command and address via the host interface (*WS*). Then the channel manager sends write command and page address (setup phase), and copies data (data transfer phase) to the NAND flash memory chip via the NAND interface (*WN*). When the data transfer ends, the address/data I/O lines of the channel manager become free. At this point, DUMBO sends an interrupt signal in spite of the incompletion of a request to maximize the utilization of the address/data I/O lines. Finally, the host system sends write confirm command (*WC*), and the corresponding NAND flash chip goes to the busy phase (*NP*).

Unlike the case of read operation, the delay time of the busy phase (*NP*) in write operation is longer than any other phases such as *WD* and *WN*. Since the host interface and the NAND interface become free during the busy phase (*NP*), we can utilize this interval to handle other requests concurrently.

4. Software architecture

In this section, we present the software architecture for the proposed NAND flash-based storage system. We also explain three optimization techniques, striping, interleaving, and pipelining, to maximize the throughput of DUMBO.

4.1. Software architecture

The software architecture for the prototype system consists of request queue management subsystem, Flash Translation Layer (FTL), and low-level device driver as presented in Fig. 5. The function of the request queue management subsystem is to rearrange I/O requests received from the operating system and to assign them to the specific channel manager. The main role of FTL is to emulate the functionality of block devices with flash memory [10]. The low-level device driver controls DUMBO according to various optimization techniques.

As described in Section 2, NAND flash memory has a restriction that the page should be erased before being rewritten in the same location. The erase operation takes much longer time in comparison with read and program operations. Therefore, an intermediate software layer called FTL is usually employed to hide the latency of erase operation as much as possible. FTL achieves this by redirecting each write request to an empty location in flash memory that has been erased in advance, and by managing the mapping information internally.

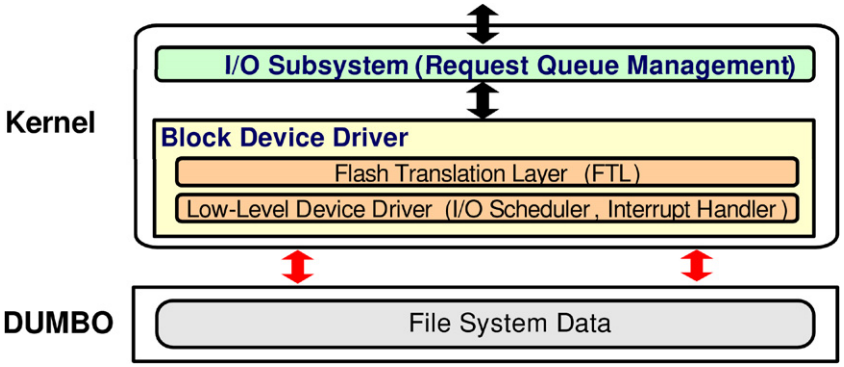

Fig. 5. The software architecture for the prototype system.

There are two main functions of FTL: address translation and garbage collection. The primary role of the address translation is to manage sector mapping information. The logical sector number of a request is translated into a physical address that represents a location of data in NAND flash memory chips. According to the granularity with which the mapping information is managed, FTL can be classified as page-mapped, block-mapped, or hybrid. The hybrid scheme uses smaller memory footprint than the page-mapped method and can program pages sequentially within a block in contrast with block-mapped method. Therefore, we implemented a hybrid FTL similar to the log-structured FTL [10] in order to reduce memory usage and to obey the sequential write order requirement in the large block NAND flash memory. Garbage collection is to reclaim free pages by erasing appropriate blocks. We use the same replacement block scheme that used for block-mapped FTL [2]. The replacement blocks are merged into a single block by the garbage collector.

4.2. Exploiting I/O parallelism

There are two kinds of I/O parallelism we can exploit. One is *intra-request parallelism* which denotes the parallel execution of a single request to reduce service time. The other is *inter-request parallelism* which indicates the parallel execution of many different requests to improve throughput.

In order to exploit intra-request parallelism, we can adopt a *striping technique*. The striping technique spreads out a request across multiple channels. Fig. 6a illustrates that the request 1 is divided into two sub-requests and those sub-requests are handled by two channels in a parallel way.

To exploit inter-request parallelism, we can think of two techniques: *interleaving* and *pipelining*. In the interleaving technique, several requests are handled in parallel by using several channel managers. For example, in Fig. 6b, two requests are processed simultaneously with two channels. The pipelining technique overlaps the processing of two requests as presented in Fig. 6c. While the interleaving technique and the striping technique occupy more than two channels, the pipelining technique occupies only a single channel.

4.2.1. Striping

We define *the striping level* to be the number of channel managers that are used to handle a single request. Those channel managers form a channel manager group. Each channel manager in the same group gets its corresponding portion of data to handle the request. The maximum striping level is four in our system because DUMBO has only four channel managers.

Let us assume that the striping level is two. A channel manager group is composed of two channel managers and DUMBO has two channel manager groups. A request is divided into two sub-requests as illustrated in Fig. 7.

In case of read operation (Fig. 7a), commands and addresses of two sub-requests are given to each channel manager sequentially (*RS*) through the host interface. Each channel manager reads data from NAND flash memory to the buffer of the channel manager concurrently (*RN*). After that, the data in

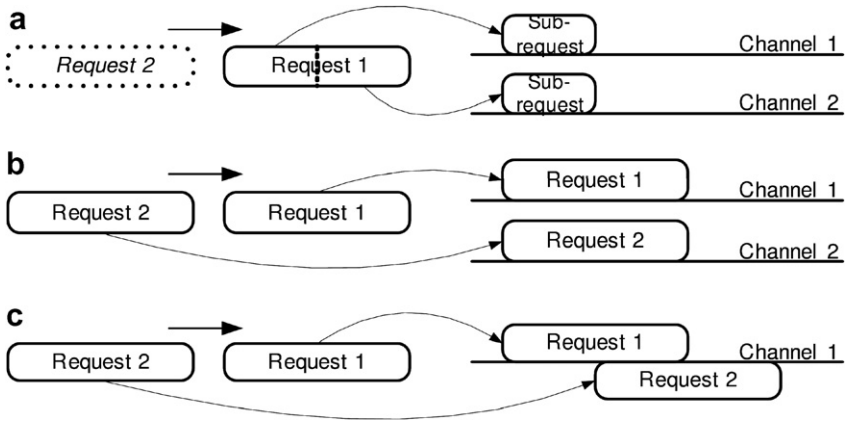

Fig. 6. Three optimization techniques to exploit I/O parallelism. (a) Striping technique; (b) interleaving technique; and (c) pipelining technique.

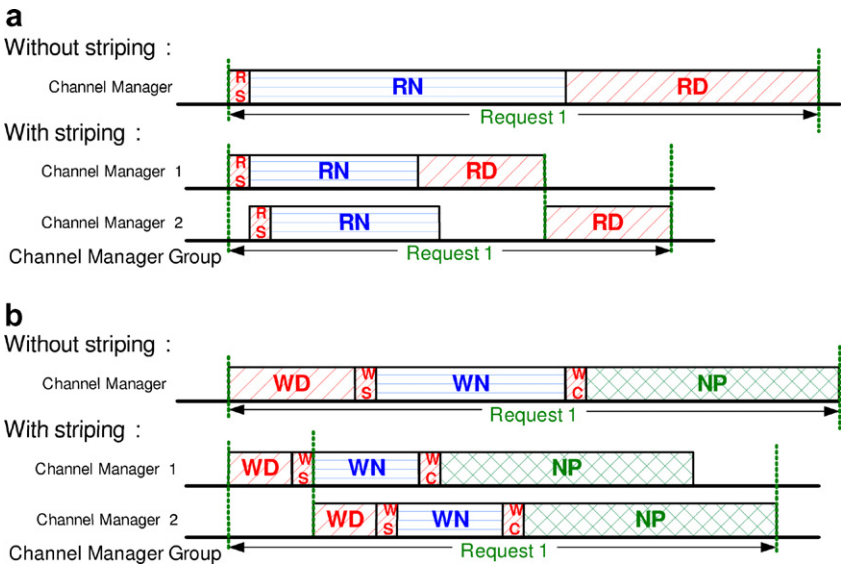

Fig. 7. The striping technique with the striping level of two. (a) Read operation and (b) write operation.

the buffers are copied to host memory sequentially (*RD*) through the host interface. The case of write operation (Fig. 7b) is similar to that of read operation except that the sequence of the data transfer phase and the busy phase is reversed.

Note that the data transfer between the host memory and DUMBO (*RS*, *RD*, *WD*, *WS*, and *WC*) is serialized via the host interface without overlapping. This is because DUMBO relies on only one host interface.

4.2.2. Interleaving

The interleaving technique exploits inter-request parallelism using multiple channel managers similar to the striping technique. The difference between interleaving and striping is the number of requests handled simultaneously. The interleaving technique handles several requests at once, while the striping technique handles only one request with several channel managers. We define *the interleaving level* to be the number of requests that can be handled

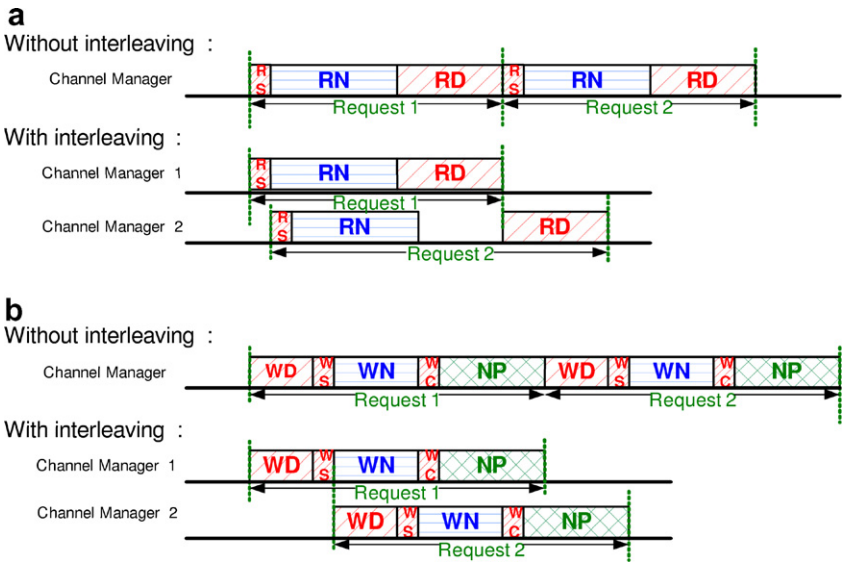

Fig. 8. The interleaving technique with the interleaving level of two. (a) Read operation and (b) write operation.

simultaneously. Again, the maximum interleaving level is four in DUMBO.

Let us assume that the interleaving level is two. In Fig. 8, request 1 and request 2 are handled concurrently by the channel manager 1 and the channel manger 2, respectively. The control order of read and write operations with interleaving are similar to that with striping except that each channel manager handles its own request separately.

For write operation, FTL needs to redirect a write request to one of the available channel managers. A simple distribution policy is a round-robin policy, where the channel manager is decided based on the logical sector number. For example, if the interleaving level is two, write requests to even-numbered sectors are redirected to the channel manager 1, while the others are to the channel manager 2. In case of read operation, however, we have no freedom to distribute incoming read requests because data are already stored in a specific location. In any case, the throughput of read or write operations can be degraded when the requests are skewed towards a specific channel manager.

4.2.3. Pipelining

The pipelining technique utilizes inter-request parallelism between the host interface and the channel manager. Since the host interface and the channel manager operate concurrently, we can send the next command to the same channel manager using the other buffer without waiting for the completion of the previous command as shown in Fig. 9. The NAND interface of the channel manager handles the previous command and the host interface deals with the next command simultaneously.

In case of read operation (Fig. 9a), when the data transfer from NAND flash memory to the buffer 1 (*RN 1*) is finished, the channel manager sends an interrupt signal to the host system. Since NAND flash memory is free at this point, we can send request 2 to the channel manager using the buffer 2 before the data in buffer 1 is copied to the host memory (*RD 1*). Therefore, the channel manager moves the data of request 2 from NAND flash memory to the buffer 2 using the NAND interface (*RN 2*) while the data in buffer 1 is transferred to the host memory using the host interface (*RD 1*).

For write operation (Fig. 9b), the data transfer of request 1 from the buffer 1 to NAND flash memory (*WN 1*) overlaps with the data copy of request 2 from the host memory to the buffer 2 (*WD 2*). Since the time of busy phase (*NP 1*) is longer than that of data transfer time (*WD 2*), we need to wait until the request 1 is completed in order to write the data of request 2 to NAND flash memory (*WN 2*, *NP 2*).

4.2.4. Putting it all together

The three techniques, striping, interleaving, and pipelining, can be combined together in our system as follows. The requests received from the operating system are assigned to a specific channel manager group according to the interleaving technique. Then

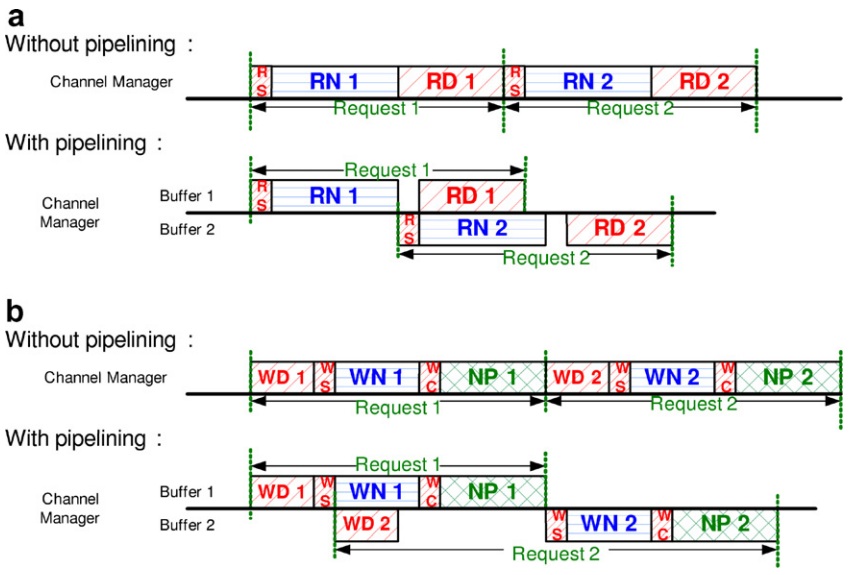

Fig. 9. The pipelining technique. (a) Read operation and (b) write operation.

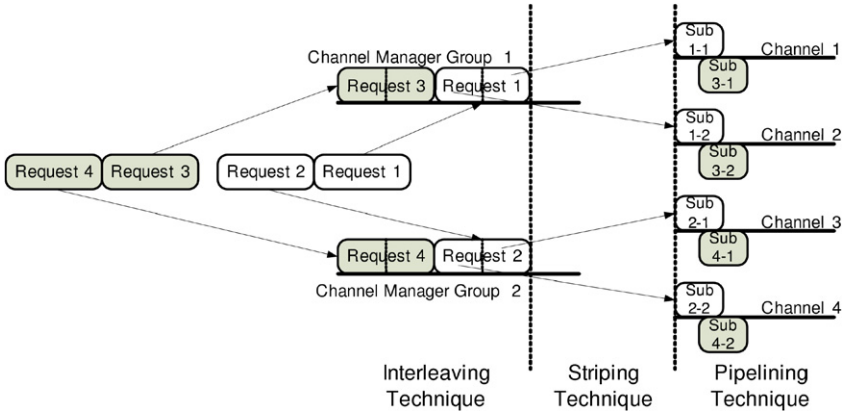

Fig. 10. The flow of requests when three optimization techniques are combined together. (It is assumed that both the striping level and the interleaving level are two.)

the request is divided into sub-requests using the striping technique. Finally, the sub-requests are processed according to the pipelining technique as presented in Fig. 10.

Since both striping and interleaving require several channel managers, we need to allocate channel managers to each technique effectively. When we apply both striping and interleaving to DUMBO, we have three possible configurations as shown in Table 1. For example, if the striping level is two, the interleaving level should be two because DUMBO has only four channel managers. The pipelining technique can be used together with striping and interleaving without any restriction because the pipelining technique does not require additional channel managers.

Another performance factor is the size of request. Generally, the large request size is desirable to maximize the throughput of storage system and to amortize the overhead associated with data transfer, although it causes more internal fragmentation. In this paper, we consider the request size ranging from 512 bytes to 8 Kbytes.

Table 1
Available configurations in DUMBO

Striping level	Interleaving level	Symbol
4	1	S4:I1
2	2	S2:I2
1	4	S1:I4

Table 2
The experimental environment

Parameters	Values
Host interface bandwidth	25 MB/s
16-bit data transfer time from NAND	50 ns[a]
Busy time of read operation	27.32 μs[a]
Busy time of program operation	196.37 μs[a]

[a] These are the values actually measured on DUMBO.

5. Performance evaluation

This section presents our experimental results. Table 2 describes our experimental environment.

5.1. Impact of optimization techniques

We first show the impact of individual optimization technique on the performance as well as the final performance when all the techniques are applied.

5.1.1. Striping

Fig. 11 presents the throughput of read and write operations under various request sizes as the striping level increases from one to four. Since the busy time of read operation is shorter than that of write operation (*NP*), the read throughput is about twice the write throughput. When the request size is 8 Kbytes, using the striping level of four improves the throughput by 73% for read and by 164% for write, compared to the case without striping. The read operation is more affected by the control overhead, which results in the smaller throughput improvement than the write operation.

The striping technique reduces only the data transfer time between DUMBO and NAND flash memory chips (*RN* and *WN*) in proportion to the striping level. The data transfer time between the

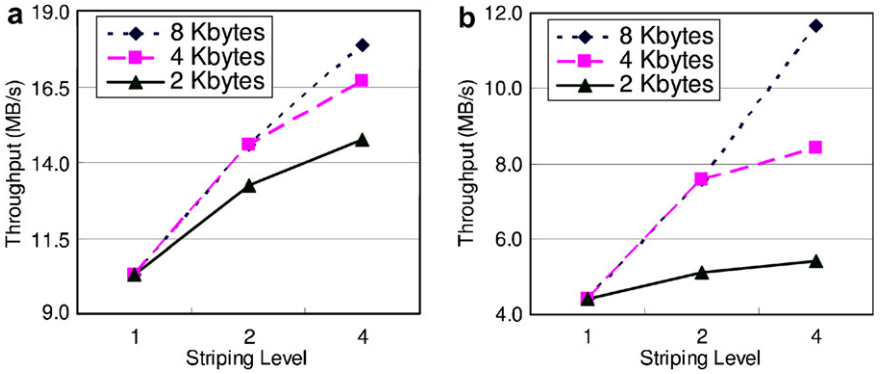

Fig. 11. The throughput under various request sizes with striping. (When the striping level is two and the request size is 4 Kbytes, two channel managers are used and a 4-Kbyte request is divided into two 2-Kbyte sub-requests.) (a) Read operation and (b) write operation.

host memory and DUMBO (*RS*, *RD*, *WD*, *WS*, and *WC*) is not improved because *RS*, *RD*, *WD*, *WS*, and *WC* must use the host interface in a mutually exclusive way. The control overhead, moreover, grows commensurate to the striping level to control more channel managers. Therefore, both the read throughput and the write throughput do not improve linearly as the striping level increases.

In Fig. 11, we can observe that increasing the striping level does not yield satisfactory throughput especially when the request size is small. This is because the size of sub-request becomes smaller than the page size of NAND flash memory, in which case partial page read (or program) operation should be performed. As we have to pay the fixed amount of time for the setup phase and the busy phase regardless of the data size, partial page read (or program) operation incurs relatively more overhead than the read (or program) operation at full page size. From these results, we can see that the striping technique is effective only when the sub-request size is equal to or greater than the page size of NAND flash memory.

5.1.2. Interleaving

Fig. 12 depicts the throughput of read and write operations under various request sizes as the interleaving level rises from one to four. When we distribute incoming requests to four channel managers, we can accomplish the throughput improvement up to 84% for read and up to 197% for write. Since commands (*RS*, *WS*, and *WC*) are sent to each channel manager via the host interface sequentially, the control overhead of channel managers is proportional to the interleaving level. Therefore, the larger interleaving level causes the smaller improvement. In addition, the bandwidth of the host interface becomes a bottleneck in the interleaving technique, because DUMBO has only one host interface.

In Fig. 12, the reason why the request size of 1 Kbytes shows consistently worse throughput than the other cases is also related to the fact that partial read or program operation incurs more overhead (cf. Section 5.1.1). The larger request size does not improve the throughput further as long as the

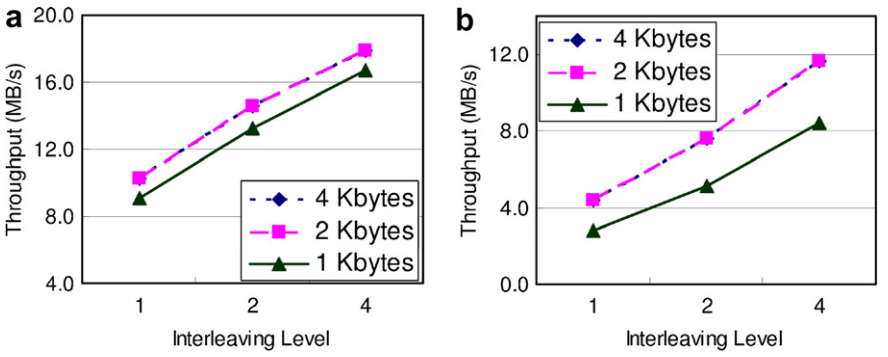

Fig. 12. The throughput under various request sizes with interleaving. (When the interleaving level is two and the request size is 2 Kbytes, two 2-Kbyte request are processed concurrently in two channel managers.) (a) Read operation and (b) write operation.

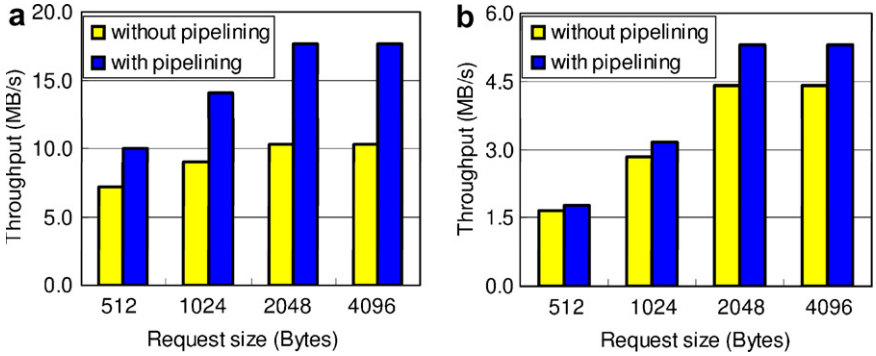

Fig. 13. The throughput improvement with pipelining. (a) Read operation and (b) write operation.

request size becomes equal to or greater than the page size of NAND flash memory.

5.1.3. Pipelining

Fig. 13 compares the throughput with and without pipelining. The use of pipelining increases the read throughput and the write throughput by 72% and by 37%, respectively. As in the interleaving technique, pipelining has no added benefit if the request size is equal to or greater than 2 Kbytes. For the smaller request size, the throughput is decreased due to the overhead of partial read and program operations.

With pipelining, the performance of NAND flash memory chips becomes a bottleneck because the bandwidth of the host interface is faster than that of NAND flash memory chip. The time using the host interface (*RS*, *RD*, *WD*, *WS*, and *WC*) overlaps with the time using the NAND interface (*RN*, *WN*, and *NP*). In case of read operation, since *RD* is not much shorter than *RN*, the pipelining technique benefits by processing *RN* and *RD* concurrently. However, the write throughput is improved slightly since the duration of *WN* and *NP* is much longer than that of *WD*.

5.1.4. Putting it all together

Fig. 14 shows the throughput of read and write operations under all possible configurations in DUMBO (cf. Table 1).

Overall, we can see that the configuration S1:I4, which maximizes the interleaving level, always performs well regardless of the request size. Recall that the striping technique splits a request into several sub-requests according to the striping level. Therefore, if the request size is small, the sub-request size can be smaller than the page size of NAND flash memory, in which case the throughput is degraded due to the partial read and program operations. As long as the sub-request size is equal to or greater than the page size of NAND flash memory, S4:I1 and S2:I2 configurations also show the performance comparable to S1:I4 configuration. This can be seen in Fig. 14, where both S4:I1 configuration with 8-Kbyte request size and S2:I2 configuration with

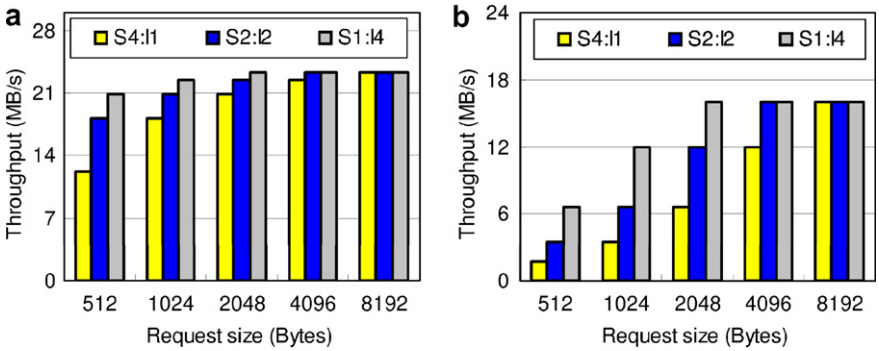

Fig. 14. The throughput under all possible configurations shown in Table 1. (a) Read operation and (b) write operation.

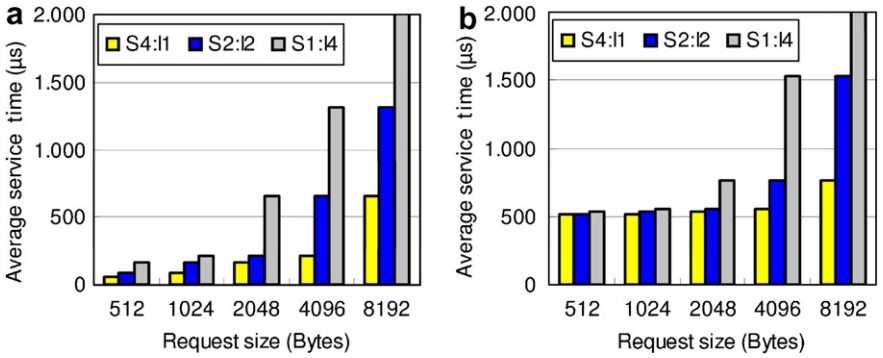

Fig. 15. The average service time under all possible configurations shown in Table 1. (a) Read operation and (b) write operation.

4-Kbyte request size exhibit the same maximum throughput.

Fig. 15 presents the average service time of read and write operations under all possible configurations in DUMBO. Contrary to the throughput, striping is very effective in reducing the average service time. This is obvious because the interleaving technique does not reduce the service time of individual requests.

From our measurement results, we selected the S2:I2 configuration with 4-Kbyte request size as the best choice for our storage system. The criteria we used to select this configuration are as follows. First, the size of sub-request with striping should be larger than the page size of NAND flash memory as stated in Section 5.1.1. Second, the request size should not be larger than 4 Kbytes because the size of default unit is 4 Kbytes in most file systems such as FAT32, NTFS, FFS, and Ext2/3 [15]. Many CPUs also use 4 Kbytes as the default memory page size. Finally, the striping level should be as large as possible to reduce the service time.

The maximum throughput of read and write operations in DUMBO with the S2:I2 configuration is 23.3 MB/s and 16.0 MB/s, respectively. This result shows that the throughput of read and write operations are improved almost 2.3 and 3.6 times, respectively, over the conventional single-channel architecture without any optimization. Since the read operation reaches the full throughput of the host interface, the improvement of read operation is slightly smaller than that of write operation.

5.2. Overall performance

We implemented the software part of the multi-channel architecture in the block device driver of Linux operating system. Fig. 16 shows the resulting throughput delivered to the user-level applications. The *raw level* in the legend indicates the maximum bandwidth of DUMBO shown in Fig. 14.

In case of read operation, the practical throughput is limited to 80% of the raw throughput. This is mainly because we use the programmed I/O in

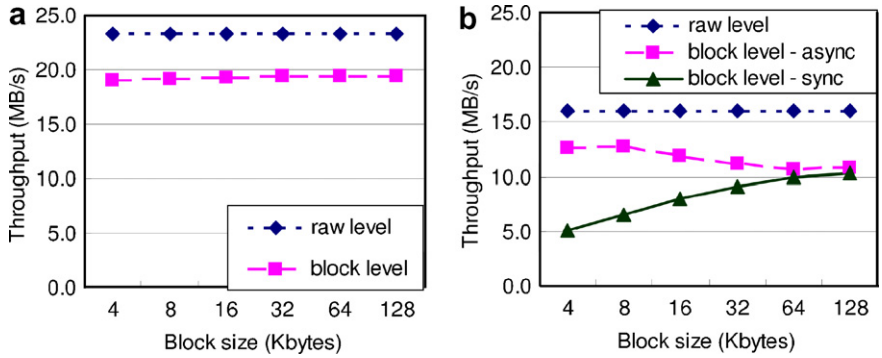

Fig. 16. The actual throughput delivered to the user-level applications. (a) Read operation and (b) write operation.

transferring data between the host memory and DUMBO. With the programmed I/O, the data transfer often interferes with the other CPU activities, such as the execution of the application code, operating system code, and the FTL code.

For write operation, Linux supports two write modes: synchronous mode and asynchronous mode. In the synchronous mode, each write() system call waits until the request is completed. Hence, the small block size cannot fully exploit the parallelism offered by DUMBO. In the asynchronous mode, however, the operating system buffers the write requests temporarily in buffer caches and issues bulk data transfer at later, which results in roughly the same throughput independent of the block size.

Unlike the read operation, the write operation shows the lower throughput as the block size increases. This is because two channel manager groups with the interleaving technique compete for the CPU to handle their own requests. Since each channel manager group waits for the end of the busy phase of program operation using busy waiting, only one channel manager group which currently acquires the CPU can handle the next request for pipelining. The other channel manager group delays the completion of the request until it holds the CPU. As the block size increases, the competition between two channel manager groups is exacerbated, thus increasing the request processing time. In addition to the competition for the CPU, the garbage collection is another factor that decreases the write performance. While write requests are processed, the garbage collector occasionally has to erase invalid blocks to make free blocks.

5.3. Overhead analysis

Table 3 shows the number of gates used by each component in DUMBO. We extracted each gate count from Synopsys design compiler [8]. One channel manager consists of 3769 gates and one or two 2112-byte (2 Kbytes + 64 bytes) buffers. For striping and interleaving, each channel manager necessitates merely one buffer, while the pipeline technique demands two buffers per channel manager. Consequently, DUMBO is made up of 17,015 gates and eight 2112-byte buffers. With this small hardware overhead, DUMBO shows 3.6 times higher overall performance compared to the conventional single-channel architecture (cf. Section 5.1.4). Note that the gate number of control logic is significantly larger than that of the NAND interface in the channel manager. This is because the control logic internally has five 32-bit registers which are used to store the address for each read or write operation, the command, and the ID and the status for a selected NAND flash memory chip.

Table 3
The number of gates used by each component in DUMBO

Host interface	Channel manager		
	Control logic	NAND interface	The size of buffer
1939[a]	3353[a]	416[a]	2112 bytes

[a] Each value is extracted from Synopsys design compiler.

The code size of software including three optimization techniques is about 4 Kbytes, when the code is compiled with gcc (GNU C Compiler) for ARM9. Although there is a little control overhead according to the number of channel managers (cf. Section 5.1), the increase of code size is negligible. The software requires only a small size of execution stack at run time, because the software does not use global data or temporary buffers.

6. Related work

To achieve high performance in flash memory-based storage system, many research efforts have been performed. Kawaguchi et al. [9] proposed a translation layer for flash file systems and studied cost-benefit policy for garbage collection. Chiang et al. [4] investigated the DAC (Dynamic dAta Clustering) scheme to cluster data during data update. On the other hand, Wu et al. [17] proposed a large non-volatile main memory storage system with write buffering in battery-backed SDRAM to hide write latency. Lee et al. [11] examined a NAND-type flash memory package with a smart buffer cache to raise the hit ratio. Recently, Park et al. [12] proposed a replacement algorithm called CFLRU (Clean First LRU) to minimize the number of write requests from the operating system. All of these previous work, however, focus on reducing the number of write or erase operations to improve the performance of flash memory-based storage system. Our work is complementary to the previous work since we are focusing on exploiting I/O parallelism for the given number of I/O requests.

In an adaptive striping architecture, Chang and Kuo [3] presented a multi-banked flash memory

storage system to utilize multiple NAND flash memory chips. Unlike DUMBO, however, their system consists of only multiple NAND flash memory chips without additional hardware controller. They just focused on an adaptive striping-aware bank assignment method which is a data placement algorithm and garbage collection technique to reduce erase operations rather than exploiting I/O parallelism on parallelized architecture. When data are updated, the method selects the specific NAND chip where the data will be written in order to evenly distribute erase operations among multiple NAND flash memory chips. The decision is generally made based on the update frequency of the data without considering hardware architecture. In fact, this is an orthogonal issue to our multi-channel architecture and we expect their method also has the similar effect on DUMBO.

7. Conclusion

This paper presents the design and implementation of high-performance NAND flash-based storage system that exploits I/O parallelism from multiple channels and multiple NAND flash memory chips. We have applied three optimization techniques to maximize I/O parallelism: striping, pipelining, and interleaving. We find out that the size of sub-request should be equal to or greater than the page size of NAND flash memory for striping, and the request size is not related to the throughput of read and write operations for interleaving. By combining all the optimization techniques carefully, our system has shown 3.6 times higher overall performance compared to the conventional single-channel architecture.

In the current implementation, only the 80% of the throughput is delivered to the user-level applications in Linux mainly due to the use of programmed I/O. To reduce the competition for the CPU in programmed I/O, we believe it would be essential to have a DMA engine. We are currently working on the second prototype system which supports more channel managers and the DMA engine.

Another cause of performance degradation in practice can be the load imbalance among channel managers in the interleaving technique. We plan to explore an algorithm that can distribute incoming requests to channel managers more effectively by investigating the characteristics of real workloads.

References

[1] G. Atwood, A. Fazio, D. Mills, B. Reaves, Intel strata flash memory technology overview, 1997. Available from: <http://www.intel.com/design/flash/isf/overview.pdf>.

[2] A. Ban, Flash file system, United States Patent No. 5,404,485, April 1995.

[3] L.P. Chang, T.W. Kuo, An adaptive striping architecture for flash memory storage systems of embedded systems, in: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02), 2002, pp. 187–196.

[4] M.L. Chiang, P.C.H. Lee, R.C. Chang, Using data clustering to improve cleaning performance for flash memory, Software-Practice and Experience 29 (3) (1999) 267–290.

[5] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, J.A. Tauber, Storage alternatives for mobile computers, in: Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI-1), November 1994, pp. 25–37.

[6] http://www.m-systems.com/site/en-US/Products/IDESCS-IFFD/IDESCSIFFD.

[7] http://www.bitmicro.com/products_enterprise_highend_ssd.php.

[8] http://www.synopsys.com/products/logic/dc_expert_ds.html.

[9] A. Kawaguchi, S. Nishioka, H. Motoda, A flash-memory based file system, in: Proceedings of the USENIX Winter Technical Conference, 1995, pp. 155–164.

[10] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, Y. Cho, A space-efficient flash translation layer for compactflash systems, IEEE Transaction on Consumer Electronics 48 (2) (2002) 366–375.

[11] J.H. Lee, G.H. Park, S.D. Kim, A new NAND-type flash memory package with smart buffer system for spatial and temporal localities, Journal of Systems Architecture 51 (2) (2005) 111–123.

[12] C. Park, J. Kang, S.Y. Park, J. Kim, Energy-aware demand paging on NAND flash-based embedded storages, in: Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED'04), 2004, pp. 338–343.

[13] C. Park, J. Seo, D. Seo, S. Kim, B. Kim, Cost-efficient memory architecture design of NAND flash memory embedded systems, in: Proceedings of the 21st International Conference on Computer Design (ICCD'03), October 2003, pp. 474–480.

[14] Samsung Electronics, 1 G × 8 bit/2 G × 16 bit NAND flash memory, 2005. Available from: <http://www.samsung.com/Products/Semiconductor/NANDFlash/SLC_LargeBlock/16Gbit/K9WAG08U1M/K9WAG08U1M.htm>.

[15] A.S. Silberschatz, P.B. Galvin, G. Gagne, Operating System Concepts, sixth ed., Wiley, New York, 2001, pp. 724–773.

[16] M. Slocombe, Samsung CEO: NAND flash will replace hard drives, September 2005. Available from: <http://digital-lifestyles.info/display_page.asp?section=platforms&id=2573>.

[17] M. Wu, W. Zwaenepoel, eNVy: a non-volatile, main memory storage system, in: Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-6), 1994, pp. 86–97.

Jeong-Uk Kang received his B.S., and M.S. degrees in Computer Science Division, Dept. of EECS from Korea Advanced Institute of Science and Technology (KAIST) in 1998, and 2000, respectively. Currently, he is enrolled in the PhD program in Computer Science Divison, KAIST. His research interests include operating systems and storage systems.

Jin-Soo Kim received his B.S., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University, Korea, in 1991, 1993, and 1999, respectively. He was with the IBM T. J. Watson Research Center as an academic visitor from 1998 to 1999. he is currently an assistant professor of the department of electrical engineering and computer science at Korea Advanced Institute of Science and Technology (KAIST). Before joining KAIST, he was a senior member of research staff at Electronics and Telecommunications Research Institute (ETRI) from 1999 to 2002. His research interests include flash memory-based storage systems and operating systems.

Chanik Park received the BS, MS, and PhD degrees in computer engineering from Seoul National University, Korea, in 1995, 1997, and 2002, respectively. He is currently a senior engineer in memory division, Semiconductor Business of Samsung Electronics. His research interests include mass storage architecture and co-design for NAND flash based embedded systems.

Hyoungjun Park received the BS degrees in electronics engineering from Yonsei University, Korea, in 2000. He was a system integration engineer at the S/W center of Samsung Electronics, Korea, from 2000 to 2003. He is currently a system integration engineer in memory division, Semiconductor Business of Samsung Electronics. His research interests include embedded systems, memory management, and FPGA prototyping.

Joonwon Lee received the B.S. degree from Seoul National University, in 1983 and the M.S. and Ph.D. degrees from the College of Computing, Georgia Institute of Technology, in 1990 and 1991, respectively. From 1983 to 1986, he was with Yugong Ltd., and from 1991 to 1992, he was with IBM research centers where he was involved in developing a scalable-shared memory multiprocessors. He is currently a faculty member at KAIST. He was a recipient of Windows NT source code. His research interests include operating systems, computer architectures, parallel processing, cluster computing and Web server.